MicroConverter™ Техническое замечание - µC003 ADuC812 MicroConverter™ как интеллектуальный сенсор в соответствии со стандартом IEEE 1451.2

Данное техническое замечание дает основные положения реализации IEEE 1451.2 совместимого интерфейса на Analog Devices ADuC812 MicroConverter. Оно определяет, как минимально реализовать стандарт IEEE 1451.2, логически приводит к разработке модульного, расширяемого и читаемого кода и рассказывает, как этот код может быть модифицирован под требования конечного пользователя.

1. ВСТУПЛЕНИЕ

1.1. ЧТО ТАКОЕ СТАНДАРТ IEEE 1451.2?

В простейшем изложении, стандарт интеллектуальных датчиков IEEE 1451.2 определяет возможность «plug-and-play» в модуле датчика, которая достигается через «электронные спецификации» («electronic data sheet»). Он определяет цифровой интерфейс для доступа к этим спецификациям, чтения данных сенсора и установки приводов («actuator»). Набор логических функций чтения и записи для доступа к электронным спецификациям датчика («transducer electronic data sheet» - TEDS) и сенсоры при этом определены. Стандартные попытки разработчиков уменьшить сложность, обычно упирались в установление связи между различными сетями и датчиками. Первичная задача стандарта – это обеспечение промышленного стандартного интерфейса для эффективного соединения датчиков с микроконтроллерами, и микроконтроллеров с сетями. Терминология, которая используется в стандарте IEEE 1451.2:

XDCR – аббревиатура «transducer», который является сенсором или приводом.

STIM – «Smart Transducer Interface Module» [рис.1], модуль интеллектуального датчика.
Сложность STIM может начинаться от одноканального сенсора или привода до многоканальных датчиков. Канал датчика назван «интеллектуальным» из-за:
- он описан как машиночитаемая TEDS
- управление и данные, связанные с каналом являются цифровыми
- обеспечиваются инициализация, чтение состояние и управление для поддержки правильного функционирования канала.

NCAP – «Network Capable Application Processor», прикладной процессор сети.
NCAP располагается между STIM и цифровой сетью, и должен обеспечивать локальную интеллектуальность. STIM подключается к сети через TII, который обеспечивает связь с NCAP.

TII – «Transducer Independent Interface», независимый интерфейс датчика
TII является 10-проводной последовательной шиной, которая также определяет:
- функцию инициализации, которая инициализирует чтение/запись датчика
- метод побитовой передачи
- протокол побайтовой записи (NCAP к STIM)
- протокол побайтового чтения (STIM к NCAP)
- передачу кадров данных

TEDS – «Transducer Electronic Data Sheet»
TEDS является спецификацией, записанной в электронном формате, который описывает STIM и датчики, связанные с ним. TEDS должна сопровождать STIM все время существования.

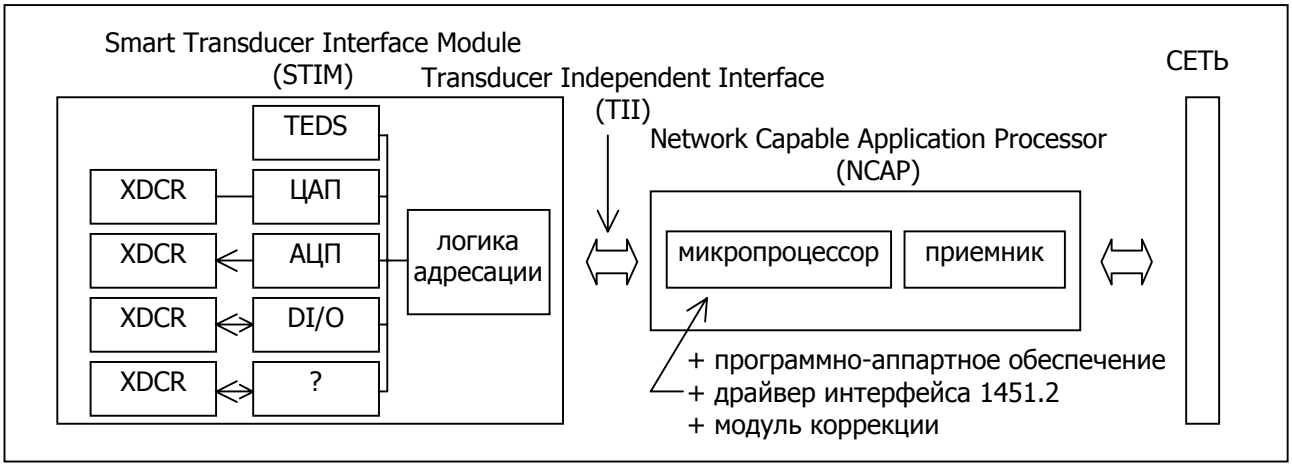

Рис.1. Обзор STIM и связи через TII и NCAP.

1.2. ВОПРОС ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ IEEE 1451.2

В общем случае, STIM включает TEDS, регистры управления и состояния, каналы датчика, маски прерываний, логику адресации и функций декодирования, функции управления передачи данных, триггер и функции подтверждения инициализации для цифрового интерфейса к TII, драйвер TII и интерфейс датчика.

TII включает передачу данных, генератор тактовой частоты, линии инициализации и подтверждения.

TEDS является электронной спецификаций. Каждый STIM имеет TEDS, которая может быть разделена на восемь разных секций:

1) Meta TEDS *[обязательная]*, мета-спецификация
   - делает доступной через интерфейс всю информацию, необходимую для получения доступа к любому каналу.
   - содержит информацию, доступную всем каналам.
   - информация постоянна и доступна только для чтения

2) Channel TEDS *[обязательная]*, спецификация каналов
   - делает доступной через интерфейс всю информацию о канале, который был адресован, для проведения правильных операций с этим каналом.
   - информация постоянна и доступна только для чтения

3) Calibration TEDS *[необязательная]*, спецификация калибровки
   - делает доступной через интерфейс всю информацию, используемую модулем коррекции при соединении с каналом, который был адресован.
   - информация может быть сконфигурирована для чтения или записи, или быть доступной только для чтения

4) Meta-Identification TEDS *[необязательная]*, мета-спецификация идентификации
   - делает доступной через интерфейс всю информацию, необходимую для идентификации STIM.
   - содержит любую идентификационную информацию, доступную для всех каналов.
   - информация постоянна и доступна только для чтения

5) Channel-Identification TEDS *[необязательная]*, спецификация идентификации канала
   - делает доступной через интерфейс всю информацию, необходимую для идентификации канала, который был адресован.
   - информация постоянна и доступна только для чтения

6) Calibration-Identification TEDS *[необязательная]*, спецификация идентификации калибровки
   - делает доступной через интерфейс информацию, описывающую калибровку STIM.
   - информация может быть сконфигурирована для чтения или записи, или быть доступной только для чтения (одинаково настроена с Calibration TEDS)

7) End User's Application-Specific TEDS *[необязательная]*, спецификация конечного пользователя, зависящая от приложения
   - содержит записываемые данные пользователя, зависящие от приложения.
   - долговременная информация.

8) Industry Extensions TEDS *[необязательная]*, спецификация расширений
   - расширение TEDS, соответствующие функции и каналы, а также расположение, тип и смысловая нагрузка данных определяется пользователем.

Программная реализация стандарта 1451.2 может быть представлена, как на рис.2. Она разделена на пять основных частей:

1) блок управления STIM и данных канала
2) интерфейс датчика STIM
3) блок TII
4) блок TEDS
5) блок адресации и функций

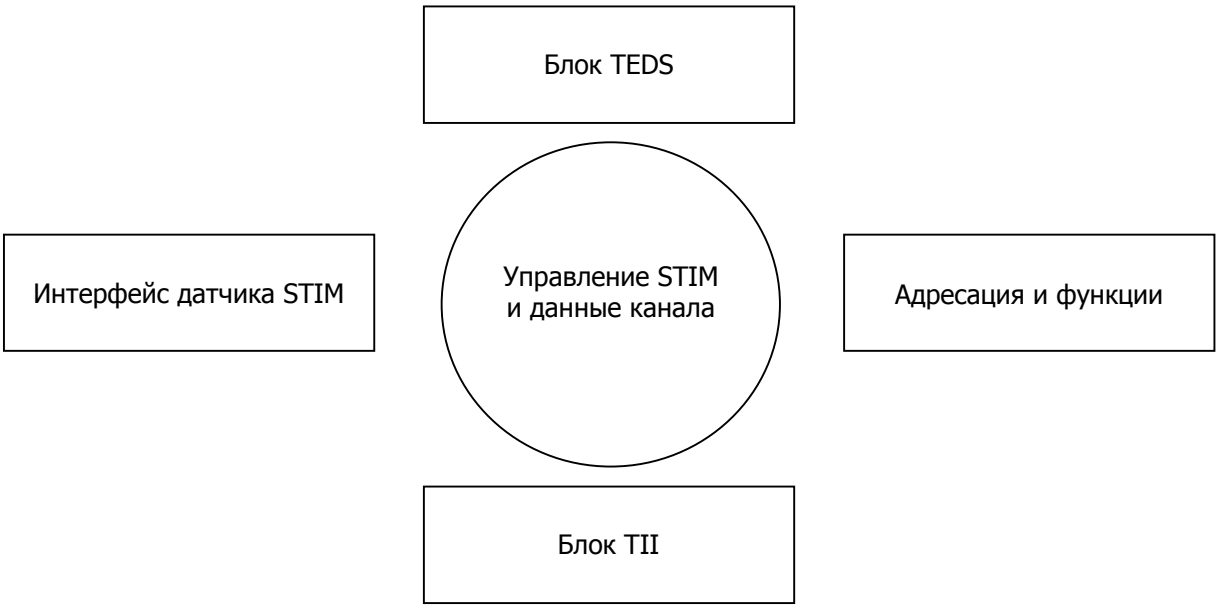

Рис.2. Стандарт 1451.2, разбитый на программные блоки.

1.3. ADUC812

ADuC812 включает ядро микроконтроллера, совместимое с 8051, 8 Кб FLASH/EE памяти программ, 640 байт FLASH/EE памяти данных, 256 байт ОЗУ, до 32 программируемых линий ввода/вывода, SPI® последовательный порт, сдвоенный ЦАП и 8-канальый 12-разрядный АЦП. Рис.3 показывает диаграмму, включающую все возможности ADuC812.

SPI порт является промышленным стандартом протокола 4-проводной синхронной последовательной связи. Он может быть сконфигурирован в режиме ведущего или ведомого, и работает от внешнего генератора тактовой частоты в режиме ведомого.

Память данных FLASH/EE, которая содержит 640 байт, сконфигурирована как 160 четырехбайтовых страниц. Интерфейс к этой памяти осуществляется через группу регистров, отображенных в области SFR (специальных функциональных регистров). Память программ FLASH/EE хранит запускаемый код пользователя.

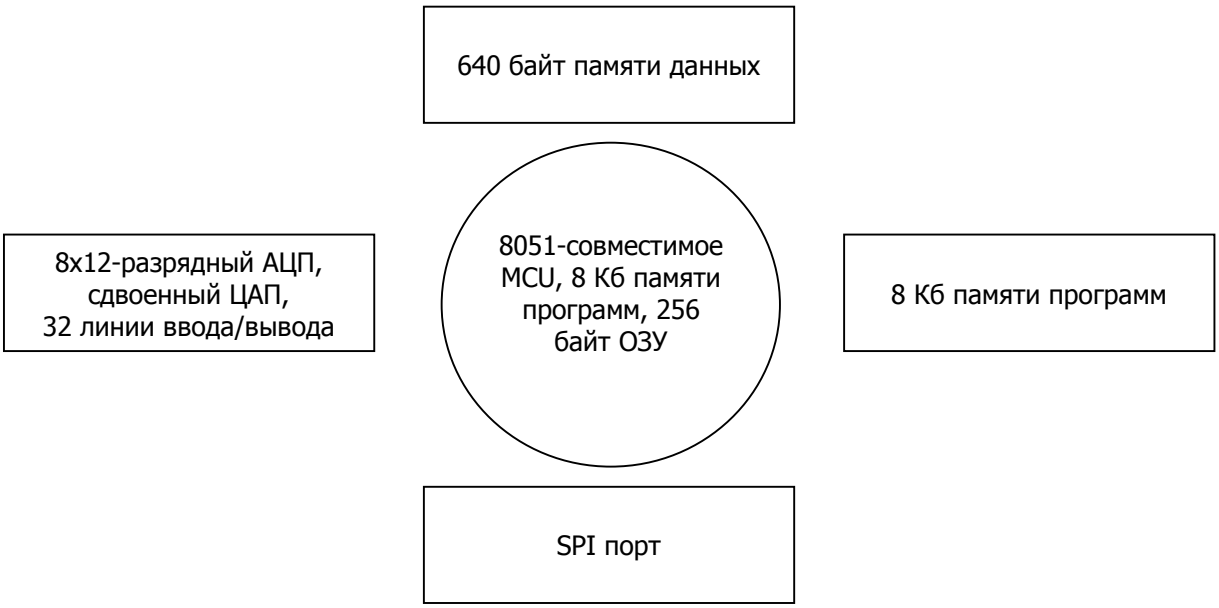

Рис.3. Диаграмма, описывающая возможности ADuC812.

1.4. СТАНДАРТ 1451.2 И ADUC812

Быстрое сравнение рис.2 и рис.3 выделяет пригодность ADuC812 как платформы для реализации STIM, основанного на стандарте IEEE 1451.2. Это показано на рис.4.

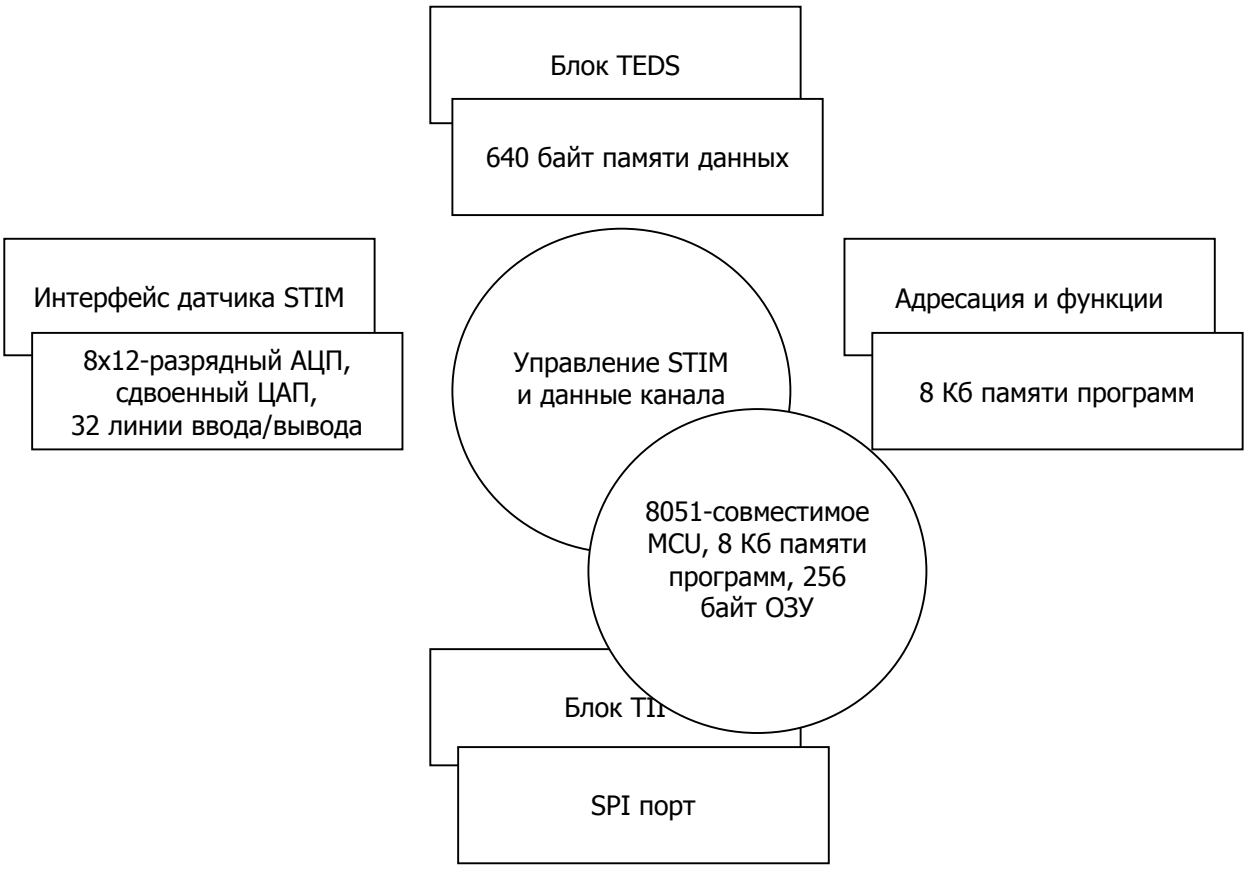

Рис.4. Перекрытие программных блоков IEEE 1451.2 возможностями ADuC812.

1) STIM будет управляться из памяти программ FLASH/EE, и данные каждого канала датчика, регистры состояния и управления будут храниться в ОЗУ все время жизни STIM.
2) интерфейс датчика будет отображен на ЦАП, АЦП и линии ввода/вывода.
3) TII будет набором SPI-порта (плюс некоторые линии ввода/вывода)
4) TEDS будет отображаться в 640 байтах FLASH/EE памяти данных.
5) и, наконец, блок адресации и функций будет размещен в FLASH/EE памяти программ.

2. РЕАЛИЗАЦИЯ

2.1. ОБЗОР ПРОГРАММНОЙ ЧАСТИ

Программа 1451.2 реализована в четырех модулях, которые отражены на рис.2. Два блока STIM объединены в один программный модуль. Модули реализованы, как описано в следующих параграфах.

2.2. ПРОГРАММНЫЕ МОДУЛИ

2.2.1. Блок STIM / Блок интерфейса датчика STIM

- *модуль назван «stim.c / stim.h»*
- *необходимы изменения по требованиям пользователя*

Этот модуль содержит определения для каналов, ассоциированных с STIM, данные, ассоциированные с каждым каналом и управляющую логику программы. Эта отдельная реализация определяет два канала: один сенсор (температурный датчик AD590) и один привод (простой вентилятор с цифровым управлением, который может быть «включен» или «выключен»). Как и модуль TII, этот модуль содержит некоторые определения, которые связаны с аппаратной частью (напр., физические линии, на которых работают сенсор и привод).

Во время изменения кода, аппаратные спецификации необходимо изменить. Программа содержит одну функцию для настройки и инициализации канала, а функция для каждого канала должна быть написана, или изменена существующая. Способ, по которому данные записываются и считываются в канале, также должен быть обсужден, хотя это является прямым добавлением/изменением в существующие функции. Наконец, тип канала определяет, как каждый канал реагирует на инициализацию. Операция инициализации должна быть адресована. Примеры, приведенные в этом техническом замечании, могут послужить хорошей отправной точкой.

2.2.2. Блок TII

- *модуль назван «tii.c / tii.i»*
- *изменения пользователя не требуются*

Модуль TII определяет физический интерфейс к NCAP. Как было замечено ранее, TII является расширенным набором SPI порта, который включен на ADuC812. Взаимодействие с физическим уровнем прописано аппаратно. Поэтому, этот модуль тесно связан с аппаратной частью. С другой стороны, смотря на функции TII через ракурс API (пользователя), связь с аппаратной частью выглядит прозрачно.

Модуль требует изменения только в том случае, когда пользователю необходимо переопределить контакты TII-интерфейса. Если пользователя устраивают существующие определения, никаких изменений не требуется.

2.2.3. Блок TEDS

- *модуль назван «teds.c / teds.h»*
- *необходимы изменения по требованиям пользователя*

Модуль TEDS описывает электронные спецификации, которые используются при реализации 1451.2. Он определяет, где отображаются TEDS, как они записаны, как они считываются, и что они содержат. Конечно, так как реализации требуют различный набор TEDS, ассоциированных с ними, модуль может потребовать изменений при написании кода. В частности, одна функция и новый начальный адрес TEDS (в памяти) должны быть добавлены для каждой TEDS.

2.2.4. Блок адресации и функций

- *модуль назван «function.c / function.h»*
- *необходимы минимальные изменения по требованиям пользователя*

Этот модуль реализует все основные функции, которые определяются стандартом. Он заботится о функциях «передачи данных», «управления», «прерывания», «чтения состояния» и «инициализации». Каждая функция предваряется тремя буквами аббревиатуры, которая показывает, к какой группе она принадлежит, напр. функция «DAT_ReadMetaTEDS()» принадлежит к группе функций «передачи данных». Модуль должен быть понятен для пользователя, и не требует больших изменений во время написания кода.

2.3. ОБЗОР АППАРАТНОЙ ЧАСТИ

Для реализации стандарта 1451.2 использовалась макетная плата ADuC812 от Analog Devices «EVAL-ADuC812QS, Rev.B01». Необходимы следующие изменения на этой плате:

- установить перемычку LK5
- повысить уровень контакта 6 на U5[1]
- соединить сенсор и привод, как показано на рис.5

Используемым NCAP является HP Bfoot 66501. HP Bfoot 66501 представляет собой законченное решение веб-сервера для разработчиков интеллектуальных датчиков и приводов. При этом оно согласовано со стандартом IEEE 1451.2 NCAP и содержит на плате 2x5 разъемов.

[1] До повышения уровня, контакт 6 на U5 управлялся Портом 3.2 [INT0, или NTRIG в этом случае], который бы проверял поступление сигнала на NTRIG от NCAP.

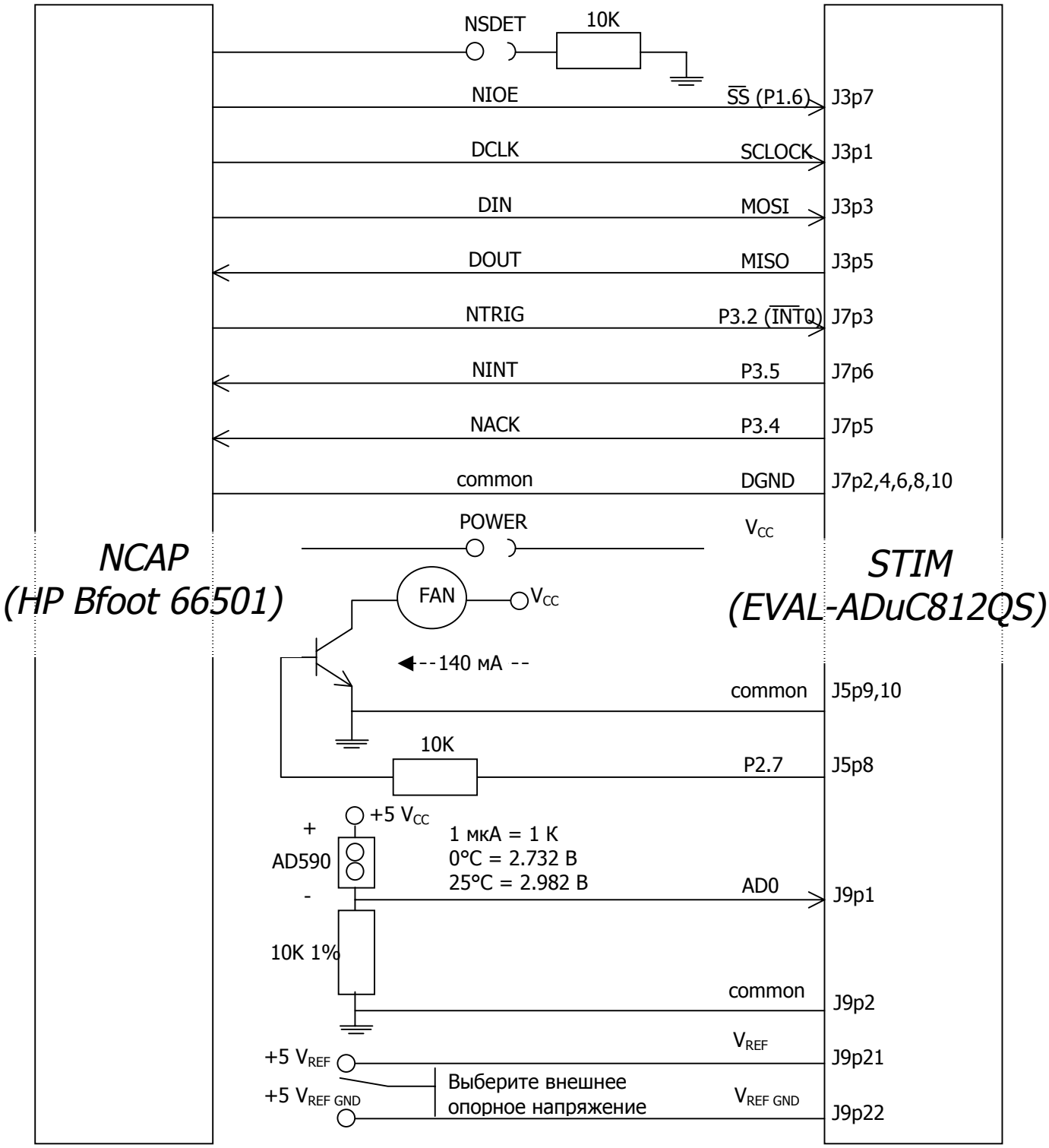

Рис.5. Аппаратное соединение платы EVAL-ADuC812QS.

2.4. ОБЩИЙ АЛГОРИТМ УПРАВЛЕНИЯ 1451.2

Алгоритм управления представлен на рис.6. Он изображает функцию «`main()`», содержащуюся в модуле STIM.

При включении платы STIM, она проходит программу инициализации. Во время этой программы:

- вся память и FLASH/EE очищаются для дальнейшей загрузки.
- TEDS загружается в область FLASH/EE память данных
- в памяти настраиваются три канала[2], ассоциированные с ними буферы данных, набор регистров, маски прерываний.
- инициализируется TII

При первой итерации основного цикла не может возникать «запрос сброса», так как этот запрос может происходить как результат функции управления, записанный в STIM через протокол передачи данных.

Передача данных проверяется на активность (`DAT_TransportActive()`), и если нет занятости, вводится блок «передачи данных». Это означает, что функции чтения и записи расшифрованы и получен корректный ответ.

При завершении передачи данных, или при отсутствии активности передачи, включается функция инициализации (`TRIG_PollTrigger()`). Триггер это способ, с помощью которого инициализируется сенсор или записывается набор данных приводу. Для порядка, в котором триггер может правильно сработать, сначала должен быть записан «адрес инициализируемого канала».

Если привод инициализирован, данные уже должны быть записаны в буфер канала, и триггер вызывает пересылку этих данных к приводу из этого буфера. В случае если инициализирован канал сенсора, данные захватываются и записываются в буфер канала за время триггера.

Данные сенсора возвращаются в NCAP только в том случае, если они запрашиваются во время передачи кадра данных. Если (во время любой активности, описанной выше) на любом из каналов обнаруживается ошибка, STIM должен установить определенный флаг в регистр состояния канала. Любой из флагов канала[3] может вызвать в STIM линию «запроса прерывания», которому NCAP будет вынужден ответить.

[2] CHANNEL_ZERO (общий канал), канал 1 (канал сенсора) и канал 2 (канал привода).
[3] Если флаг состояния разрешен в прерывании канала или внешней маске прерываний.

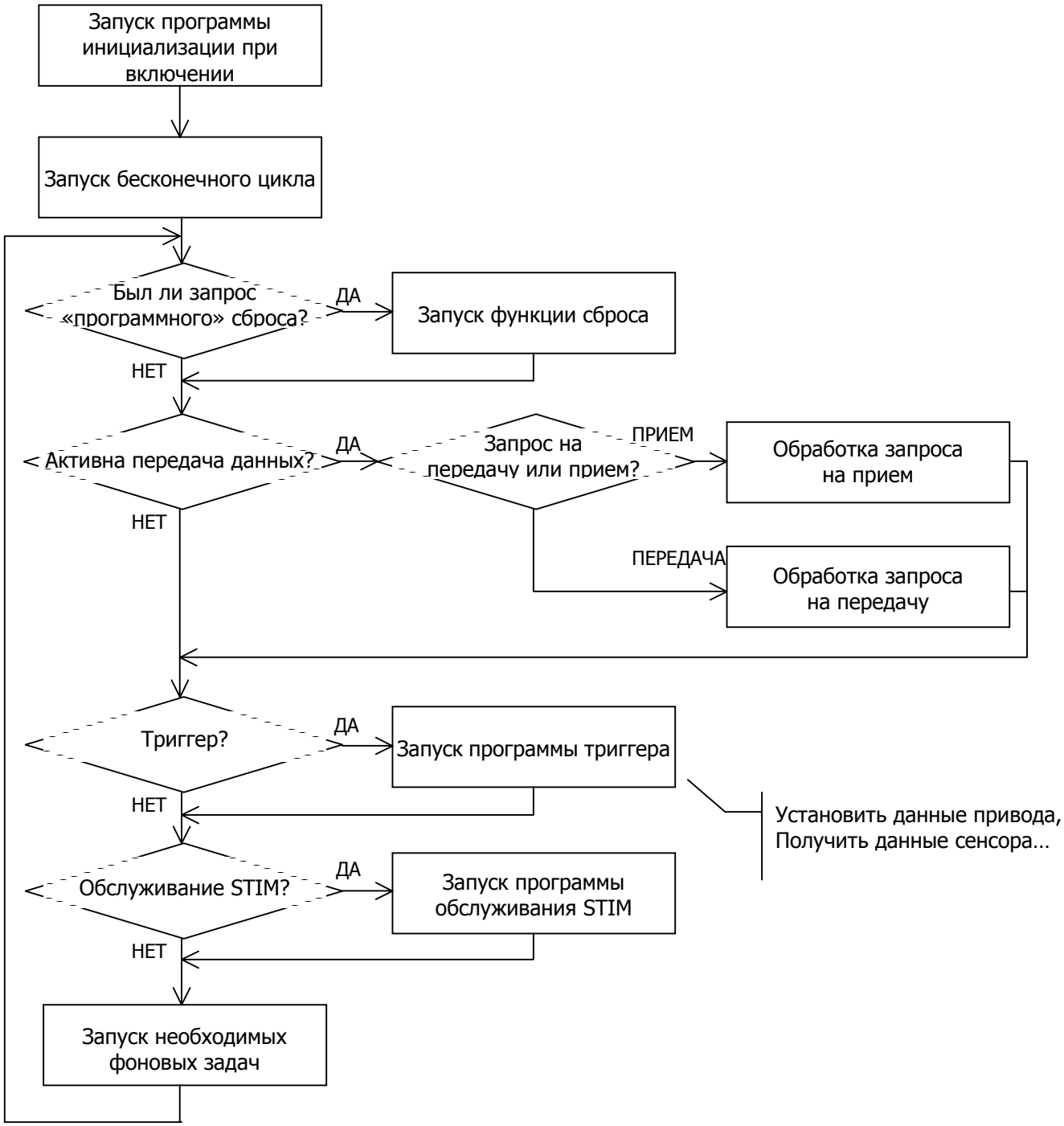

Рис.6. Блок-схема алгоритма управления 1451.2.

2.5. КОМПОНОВКА КОДА

Код может быть скомпонован с использованием среды разработки Keil:

1) Запустите программу Keil uVision-51
2) Откройте новый проект – «New Project». Назовите его «1451.prj»
3) Перейдите в каталог, в котором хранятся исходные тексты программы, и выберете следующие файлы:
   - function.c
   - teds.c
   - tii.c
   - stim.c
4) Выберете «Open All» и далее «Save»
5) Перейдите в настройки – «Options»-«BL51 Code Banking Linker» и выберете страницу «Size/Location». Измените значение «RAM Size» на 256 байт. Нажмите «OK».
6) Нажмите «Alt-F8» для создания hex-файла.
7) Теперь должен быть файл с именем «1451.hex» в том же каталоге, что и исходный код.

2.6. ПРОГРАММИРОВАНИЕ МАКЕТНОЙ ПЛАТЫ

1) Соедините через RS-232 кабель порт компьютера COM1 и макетную плату EVAL-ADuC812QS.
2) Установите перемычку LK3
3) Включите питание и нажмите «RESET»
4) Запустите программу последовательной загрузки (находится в каталоге ADuC812)
5) Введите полный путь и имя файла «1451.hex»
6) Когда программа последовательной загрузки завершит свою работу (это займет несколько секунд), снимите перемычку LK3 и нажмите «RESET» снова. Вентилятор должен повернуться (немного) при инициализации.
7) Отключите питание.
8) Вы готовы соединить макетную плату к NCAP через TII.

2.7. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Мы использовали Hewlett Packard Bfoot 66501, промышленную Ethernet плату, для запуска тестов и соединения с STIM реализацией.

HP Bfoot 66501 является законченным решением веб-сервера для разработчиков интеллектуальных датчиков и приводов, и включает также поддержку стандарта IEEE 1451.2 NCAP. Оно содержит на плате 2x5 разъемов и может обеспечить возможность запуска 1451.2 транзакций передачи данных и отображать результаты на любом веб-броузере. На плате расположен апплет, позволяющий автоматически инициализировать любой подключенный сенсор и графически отображать результаты в реальном времени.

3. ИЗМЕНЕНИЕ КОДА ПО ТРЕБОВАНИЯМ ПОЛЬЗОВАТЕЛЯ

Исходный код, рассмотренный в этом техническом замечании, может быть загружен с веб-сайта Analog Devices http://www.analog.com/microconverter

3.1. ДОСТУПНЫЕ ИСХОДНЫЕ ФАЙЛЫ

Доступные исходные файлы описаны в таблице.

Таблица 1. Доступные исходные файлы.

Имя файла	Описание	Доступные изменения
`stim.c`	Содержит определения для размещения сенсора и привода, и методов доступа к ним. Он также определяет экземпляр типа «канал» для каждого реализованного канала. Модуль обеспечивает функции для доступа к каждому каналу и привязанных к нему данных. Определены программа инициализация канала, информация по версии STIM и программа сброса STIM. Описана функция `main()`.	Изменения необходимы для отражения вашей версии STIM. Этот модуль, больше чем остальные, должен изменяться с осторожностью. Основная структура не должна изменяться, но могут быть изменены определения каналов, их инициализация и способ управления данными.
`tii.c`	Содержит физическое определение «интерфейса независимого датчика». Отражает интерфейс на аппаратную часть SPI и контакты порта ввода/вывода. Обеспечивает независимые от аппаратной части функции определения для TII интеграции. Описывает программу прерывания для управления данными через SPI.	В этом модуле не требуются изменения, если пользователь работает с примененным физическим отражением интерфейса. Если нет, то необходимо изменить только физические определения ввода/вывода.
`teds.c`	Содержит определение TEDS, метод настройки TEDS и записи в долговременную память.	Почти все TEDS для всех реализаций будут разными. Содержание TEDS, число определенных TEDS, отображение TEDS в памяти и способ, по которому записываются TEDS, должны быть изменены.
`function.c`	Содержит все стандартные функции 1451.2. Они включают передачу данных, инициализацию, опрос состояния, прерывание и управление.	Требуются очень небольшие изменения. Лучше оставить этот модуль как есть.[4]
`stim.h`	Содержит определение для общей структуры «данных датчика». XDCR_DATA (требуется в CHANNEL_ZERO). Содержит определение типа CHANNEL, являющимся экземпляром для каждого канала.	Изменения должны отражать любые изменения в файле «`stim.c`».Могут потребоваться изменения в структурах XDCR_DATA и CHANNEL.
`tii.h`	Содержит прототипы функций, реализованных в файле «`tii.c`»	Изменения должны отражать любые изменения в файле «`tii.c`».

[4] В этой версии кода:
- функции передачи данных, которые записывают «calibration» и «calibration id» TEDS не работают.
- Функции передачи данных для чтения и записи данных датчика поддерживают только два канала.

Таблица 1. Доступные исходные файлы. (продолжение)

teds.h	Описывает число используемых каналов, CHANNEL_ZERO как константу и «data model length» для каждого канала. Определяет структуру для Meta- и Channel-TEDS (других TEDS в этой версии нет). Включены прототипы функций.	Реализовывается, зависящее от модели STIM, число TEDS, число каналов и структуры TEDS. Изменения должны отражать любые изменения в файле «teds.c».
function.h	Содержит все определения, определения типов и т.п. соответствующие «function.c». Содержит все прототипы функций и описания каждой группы функций, требуемых стандартом IEEE 1451.2	Изменения должны отражать любые изменения в файле «function.c».
datatype.h	Определяет типы данных по стандарту IEEE 1451.2 секции 3.3. Содержит множество общих типов данных (NULL, boolean, NaN32), используемых при реализации стандарта 1451.2	Не требует изменений
aduc812.h	Содержит определения SFR регистров, используемых в ADuC812.	Не требует изменений

3.2. ПРИМЕР ИЗМЕНЕНИЯ КОДА

Приведем пример, как должен быть изменен загруженный исходный код для отражения STIM, который содержит два сенсора (каналы АЦП 0 и 1) и два цифровых привода (контакты порта 2.6 и 2.7). Мы расположили первый сенсор STIM на логическом канале 0, а второй сенсор на логическом канале 3. Приводы будут расположены на логических каналах 2 и 4.

3.2.1. «stim.c» и «stim.h»

«stim.h»

- «Ch3» и «Ch4» должны быть добавлены в определение XDCR_DATA. Так как они являются сенсором и приводом, их тип данных будет «unsigned int» и «boolean» соответственно. Тип данных для «ChData» в этом примере не изменяется. Если вводится тип данных больший, чем любой существующий, «ChData» примет этот тип.

- необходимо добавить прототипы функций для STIM_InitCh3, STIM_InitCh4, STIM_GetCh3Sample и STIM_SetCh4State.

«stim.c»

- необходимо определить второй привод на контакте 2.6

```
sbit FAN_CH2 = P2 ^ 7;      // на канале 2
sbit RELAY_CH4 = P2 ^ 6;    // на канале 4
```

- в STIM_GetXdcrData() добавить две ветки для новых каналов.

```
case 3:*(unsigned int*)pBuf = maChannelData[ucChNum].Data.Ch3;
break;
case 4: *pBuf = maChannelData[ucChNum].Data.Ch4;
break;
```

- добавить две функции инициализации каналов, STIM_InitCh3 и STIM_Init4. Используйте функции STIM_InitCh1 и STIM_Init2 как шаблон.

- добавить функцию STIM_GetCh3Sample, и еще одну STIM_SetCh4State. Опять же, функции STIM_GetCh1Sample и STIM_SetCh2State могут быть использованы как шаблон.

- в функции «main»
  (Блок инициализации)
  - добавить вызовы функций STIM_InitCh3 и STIM_InitCh4.
  - добавить вызовы функций TEDS_SetupCh3Teds и TEDS_SetupCh4Teds.
  - добавить вызовы в INT_SetInterruptMask для каналов 3 и 4, для дополнительной и стандартной маски прерываний.

  (Блок триггера)
  - добавить ветки, описывающие действия при событии триггера для каналов 3 и 4.

3.2.2. «teds.c» и «teds.h»

«teds.h»
- изменить определение для «NUM_CHANNELS» на 4.

- добавить два определения для CH3_DATA_MODEL_LENGTH и CH4_DATA_MODEL_LENGTH, присвоив им «2» и «1» соответственно.

- добавить прототипы функций для TEDS_SetupCh3Teds и TEDS_SetupCh4Teds.

«teds.c»
- добавить два новых канала в область память TEDS, назначив им адреса.

```
#define CH3_TEDS_ADDRESS 0x43
#define CH4_TEDS_ADDRESS 0x5B
```

- добавить две новые функции настройки TEDS, для каждого из новых каналов (TEDS_SetupCh3Teds и TEDS_SetupCh4Teds). Эти функции могут быть основаны на существующих, при изменении только содержимого структуры данных TEDS. Также должен быть изменены внутренние ссылки на номер канала и/или адрес TEDS.

- в функции TEDS_GetTEDSHandle добавить две новых ветки для новых каналов TEDS.

3.2.3. «function.c» и «function.h»

«function.h»
- изменения не требуется.

«function.c»
- в функции DAT_ReadXdcrData добавить две ветки:

```
else if(ucChan==3) DataLength = CH3_DATA_MODEL_LENGTH;
else if(ucChan==4) DataLength = CH4_DATA_MODEL_LENGTH;
```

- добавить аналогичные строки в конструкцию «switch…case» в DAT_WriteXdcrData.

3.2.4. Оставшиеся модули

«tii.c», «tii.h» , «datatype.h» , «aduc812.h»
- изменения не требуется.

4. ЗАКЛЮЧЕНИЕ

Главной задачей этого IEEE 1451.2 приложения являются:
- создать минимальную стандартную реализацию
- создать модульный, расширяемый и читаемый исходный код
- сделать удобным изменение кода

Пример, приведенный в части 3 (и в приложении 2) показывает, как существующие модули исходного кода могут быть изменены и расширены для удовлетворения требований пользователя. Некоторые моменты нужно знать во время изменения кода:

ОЗУ:
Имеется 256 байт, расположенной на плате, ОЗУ. Из них:
- 100 байт зарезервированы для загрузки TEDS
- 10 байт зарезервированы для каждого внедренного канала
- пространство данных (во время выполнения) занимает больше 30 байт.
- пространство стека (во время выполнения) занимает от 20 до 30 байт.

Память данных FLASH/EE:
Имеется 640 байт, расположенной на плате, памяти данных FLASH/EE.
- 76 байт используется минимальной версией Meta-TEDS
- 96 байт используется для каждой минимальной версии Channel-TEDS.
- Другие версии TEDS не определены по размеру, но могут быть больше.
- До пяти Channel-TEDS (+Meta-TEDS) могут быть расположены в области памяти данных FLASH/EE.

Физическая реализация:
Интерфейс TII коннектора (10-проводной плоский кабель), должен быть как можно короче. Для надежных операций, рекомендуется применять кабель до нескольких дюймов.

Потребление ресурсов:

Таблица 2. Свободные и занятые ресурсы

Оригинальный код: вкл. 2 Channel-TEDS, 1 Meta-TEDS		
Ресурс	**Занято**	**Свободно**
256 байт ОЗУ	178 байт	78 байт
640 байт FLASH/EE данных	268 байт	372 байт
8 Кб FLASH/EE программ	5,534 байт	2,657 байт
Код примера: вкл. 4 Channel-TEDS, 1 Meta-TEDS, 1 Meta-ID TEDS		
Ресурс	**Занято**	**Свободно**
256 байт ОЗУ	197 байт	59 байт
640 байт FLASH/EE данных	552 байт	88 байт
8 Кб FLASH/EE программ	6,553 байт	1,638 байт

Если требуется большее число TEDS, они могут быть расположены в FLASH/EE памяти программ. Заметьте, что чем больше требуется TEDS, тем больше будет код программы, поэтому имеется предел количества TEDS, расположенных в этой области.

Альтернативой может быть использование внешней EEPROM для хранения TEDS. В этом случае, функции чтения и записи TEDS должны быть написаны заново, изменится также и схема распределения памяти.

В таблице 2 не приведены требования по расположению стека в ОЗУ. Стек будет использовать верхнюю область памяти ОЗУ. Число требуемой ОЗУ будет различным, и заключается между 20 и 30 байтами. Поэтому, число доступной памяти ОЗУ, приведенной в таблице не совсем корректно.

5. ДАЛЬНЕЙШАЯ ПОДДЕРЖКА ПРИЛОЖЕНИЙ ПО РЕАЛИЗАЦИИ 1451.2

Реализация, описанная в этом техническом замечании, основана на полной версии стандарта IEEE 1451.2, и разработана PEI Technologies, университет Лимерика, Ирландия.

PEI Technologies специализируется на исследовании, создании и разработке аппаратных и программных средств для распределенных систем управления, основанных на промышленных стандартных последовательных протоколах связи.

PEI Technologies доступны для поддержки любой реализации IEEE 1451.2, или других стандартов серии 1451 (например, 1451.1 поддержка NCAP для протоколов Ethernet, CAN и LonWorks). Посетите сайт http://www.ul.ie/~pei для дополнительной информации.

6. ССЫЛКИ

«Sensors Smarten Up» EDN Cover Story, Bill Travis, Март 1999
«Smart sensor standart looks set for takeoff» EE Times, Terry Costlow, Октябрь 1998
«IEEE 1451.2-1997» IEEE, Сентябрь 1998

ПРИЛОЖЕНИЕ 1

ОБЗОР МОДЕЛИ

Схема распределения памяти

Рис.7 показывает программную модель ADuC812. ADuC812 разделяет адресное пространство для памяти программ и памяти данных. Дополнительные 640 байт FLASH/EE имеются в распоряжении пользователя посредством косвенного доступа через группу управляющих регистров в пространстве SFR памяти данных. Нижние 128 байт памяти имеют прямой доступ, в то время как верхние 128 байт могут быть доступны через косвенную адресацию.

Пространство SFR доступно только через прямую адресацию и обеспечивает интерфейс между ЦП и встроенной периферией.

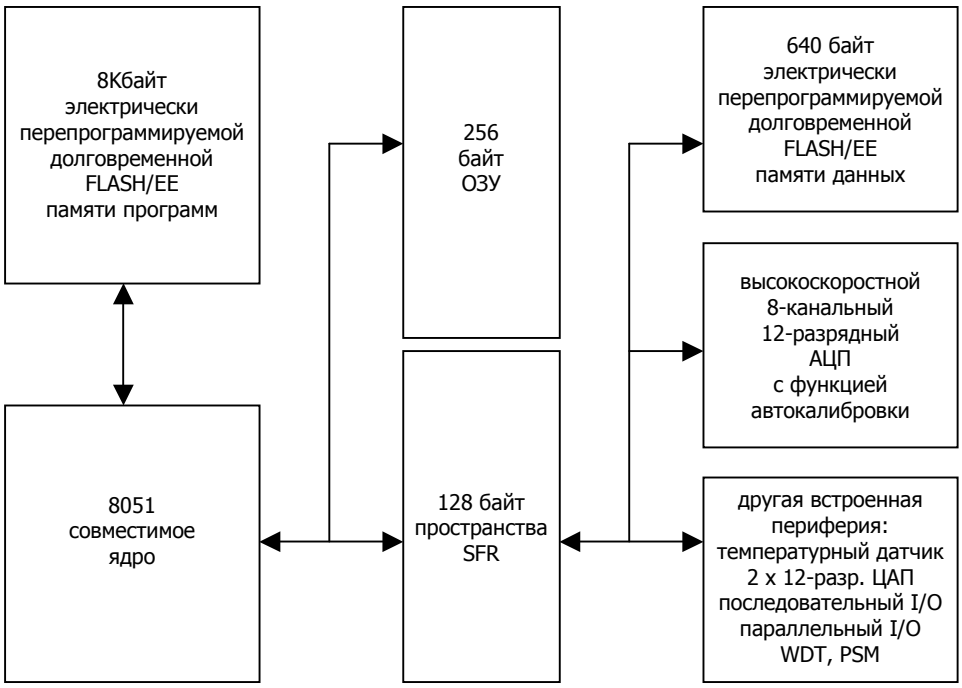

Рис.7. Программная модель ADuC812.

Рис.8 показывает, как реализация 1451 использует программную модель ADuC812. TEDS расположены в пространстве 640 байт FLASH/EE памяти данных, TII и привод напрямую включены в блок «периферия», а сенсор расположен в блоке АЦП. Все эти возможности доступны и контролируются через пространство SFR.

Стандартная память ОЗУ используется для хранения данных канала датчика STIM и регистров. Она должна зарезервировать буфер, который может быть индивидуально загружен TEDS, и из которого TEDS может считывать данные из FLASH/EE памяти. При этом ОЗУ должна хранить «локальные» и «системные» переменные, которые нужны программе, а также разместить область стека во время выполнения. Все эти требования накладывают ограничения на размер буфера TEDS, который пользователь должен контролировать.

Конечно, все запрограммированные функции хранятся в FLASH/EE памяти программ.

Пунктирная линия на Рис.8. показывает логическую связь между функциями `DAT_Write.x.TEDS()` и `DAT_Read.x.Teds()`, и существующие чтение и запись от/в пространство TEDS FLASH/EE памяти данных. Эти вызовы `TEDS_` функций разработаны, чтобы быть логически понятными для пользователя, при этом метод реализации не важен.

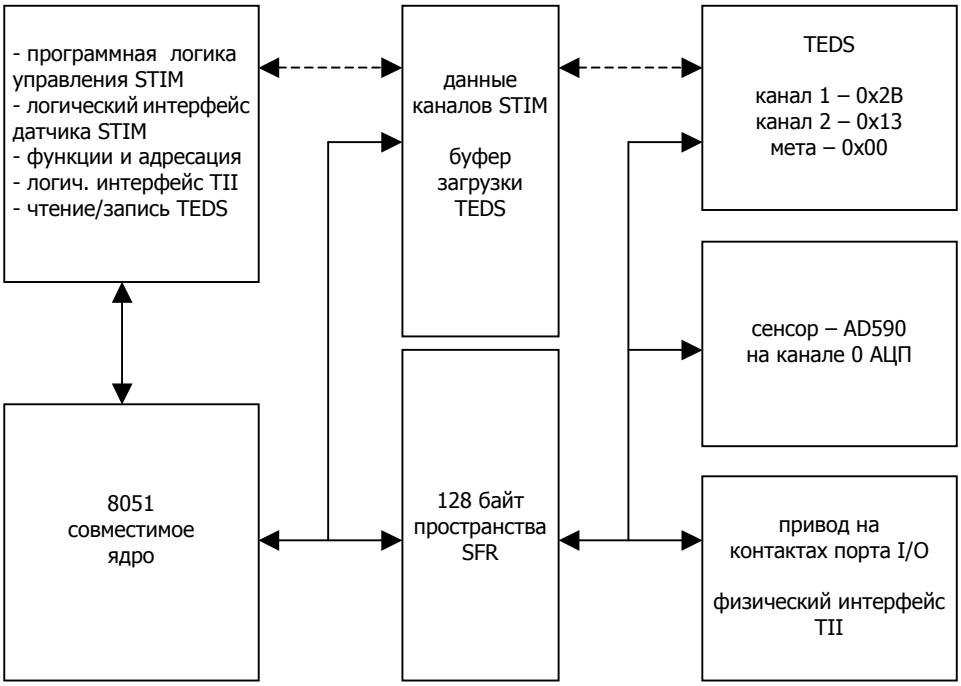

Рис.8. Реализация, отраженная на программной модели ADuC812.

ДЕТАЛИ МОДЕЛИ

Детали TII и соединения датчика показаны ранее на Рис.5.

1. TII

Интерфейс TII описывает 10 линий, названные:

Таблица 3. Определение TII

Группа	Линия	Сокращение	Описание
Данные	DATA_OUT DATA_IN DATA_CLOCK N_IO_ENABLE	DOUT DIN DCLK NIOE	Передача данных от STIM к NCAP Передача адреса и данных от NCAP к STIM Положительный перепад, фиксирующий данные на линиях DIN и DOUT Сигнал, показывающий активность передачи данных и определяющий формирование кадров данных
Инициализация	N_TRIGGER	NTRIG	Выполняет функции инициализации
Поддержка	POWER COMMON N_ACKNOWLEDGE N_STIM_DETECT	POWER COMMON NACK NSDET	Номинальные 5В источника питания «Земля» Обеспечивает две функции: 1) Подтверждение триггера 2) Подтверждение данных Используется NCAP для определения присутствия STIM
Прерывание	N_IO_INTERRUPT	NINT	Используется STIM для запроса обслуживания у NCAP

Линии, определенные в таблице 3, реализованы в соответствии с таблицей 4:

Таблица 4. Существующая реализация TII на EVAL-ADuC812QS

Сокращение	Контакт ADuC812	EVAL-ADuC812QS	Описание
DOUT	MISO	J3 контакт 5	контакт «Master In, Slave Out» часть интерфейса SPI порта
DIN	MOSI	J3 контакт 3	контакт «Master Out, Slave In» часть интерфейса SPI порта
DCLK	SCLOCK	J3 контакт 1	Последовательная синхронизация часть интерфейса SPI порта
NIOE	выбор ведомого	J3 контакт 7	контакт «выбор ведомого» часть интерфейса SPI порта если используется, то показывает, что SPI находится в режиме ведомого, т.е. используется внешняя синхронизация
NTRIG	порт 3.2 (INT0)	J7 контакт 3	общий контакт, сконфигурированный как вход. Уместно соединить NTRIG с этим контактом, так как он может быть настроен (программой) как внешняя линия прерывания[5]

[5] Хотя 1451 лучше реализуется без прерывания по NTRIG, могут быть причины, когда этот вариант мог бы быть более выгодным.

POWER	V_{CC}	шина питания	Питание для STIM – по определению TII, должен снабжаться от NCAP.
COMMON	GND	шина «земли»	Шина заземления для STIM
NACK	порт 3.4	J7 контакт 5	Контакт порта общего назначения, сконфигурированный как выход.
NSDET	нет	через нагруз. резистор к земле	Линия NSDET находится на активном низком уровне напряжения, поэтому на стороне STIM линия должна быть заземлена.
NINT	порт 3.5	J7 контакт 6	Контакт порта общего назначения, сконфигурированный как выход.

Для совместимости с TII, SPI должен быть инициализирован в режиме ведомого при помощи установки синхросигнала ждущего режима на высокий уровень, и установки синхронизации для защелки по нарастающему фронту (см. IEEE 1451.2, секция 6.2 и 6.3.7)

Это достигается записью в SFR регистры записью *SPICON* и *SPE* и разрешением SPI-прерывания (через *IE2*):

```
SPICON = 0x0C;     // режим ведомого: CPOL=1, CPHA=1, SPIM=0
IE2 = 0x01;        // разрешить прерывание SPI
SPE = 1;           // разрешить SPI
```

Линии TII, которые используют контакты порта (напр. NINT, NACK, NTRIG, NIOE[6] и др.) должны быть сконфигурированы (как вход или выход) в начале модуля. Их физическое отражение определяется вначале:

```
// Входы STIM...
//

sbit NIOE = P1 ^ 5;
sbit NTRIG = P3 ^ 2;

// Выходы STIM...
//

sbit NACK = P3 ^ 4;
sbit NINT = P3 ^ 5;
```

и их направление указывается в функции `TII_Initialise()`:

```
// Настройка Port 1.5 как цифровой вход (NIOE pin, 'SS')
// - это устанавливается записью 0 в этот разряд SFR[7].
P1 = P1 & 0xDF;

// Настройка Port 3.4 и 3.5 как цифровой выход (NACK и NINT)
// - это устанавливается записью 0 в эти разряды SFR.
// Настройка Port 3.2 как цифровой вход (NTRIG)
// - это устанавливается записью 1 в этот разряд SFR.

P3 = P3 & 0xCF;
P3 = P3 | 0x04;
```

[6] Заметьте, что NIOE использует линию «выбор ведомого» - вторую функцию порта 1.5. «Выбор ведомого» настраивается записью «0» в порт 1.5, после этого порт 1.5 показывает состояние «выбор ведомого»

[7] см. спецификацию ADuC812

2. Сенсор и привод

Сенсором является AD590 (температурный датчик), он является входом канала 0 АЦП ADuC812. Он определяется и инициализируется в функции STIM_InitChl()

```
// Настройка Port 1.0 как аналоговый вход (вход сенсора на ADC0)
// - это устанавливается записью 1 в этот разряд SFR.
//

P1 = P1 | 0x01;
ADCCON1 = 0x6C; // ADCCON1 01101100
ADCCON3 = 0x00; // ADCCON3 00000000
ADCCON2 = 0x00; // Не разрешать любой режим преобразования данных (сейчас)
```

Каждый раз, когда требуются[8] данные с канала 1, происходит единичное преобразование. Когда цикл заканчивается, доступна выборка данных. Преобразование данных должно ждать переключения бита прерывания АЦП перед тем, когда выборка может считаться правильной. См. `STIM_GetChlSample()`.

```
// Установите бит 'single sample' в ADCCON2 для инициализации
// цикла единичного преобразования
//

SCONV = 1;

// Ожидание прерывания АЦП (данные получены) ...
//

while( !ADCI );

<< Код, расположенный здесь (не показан), обнаруживает и фиксирует выборку
данных. Когда операция заканчивается, очистите бит единичного
преобразования и бит прерывания АЦП, после чего АЦП готов к следующей
выборке данных. >>

SCONV = 0;
ADCI = 0;
```

Приводом является простой вентилятор, который контролируется контактом 2.7. Он определен в начале модуля «stim.h», и расположен на канале 2 1451.2. Привод инициализируется в функции STIM_InitCh2() и управляется из STIM_SetCh2State()

```
// Определение контакта, который управляет приводом (напр. вентилятором)
// на канале 2.
//

sbit FAN_CH2 = P2 ^ 7;
```

[8] Запрос данных приходит в форме «триггера», который определяется NTRIG.

3. TEDS

В первую очередь, необходимо определить, какое количество TEDS требуется, и где они должны располагаться. Единственными обязательными TEDS являются Channel- и Meta-TEDS. Это означает, что необходимо описать по одной TEDS для каждого канала (Channel 1 TEDS и Channel 2 TEDS) и TEDS, которая описывает систему в целом (Meta TEDS). В этой реализации не описываются другие TEDS.

Meta TEDS для этой системы занимает 76 байт, и 96 байт занимают каждая из Channel TEDS. Они отображаются в 640 байтах FLASH/EE памяти данных.

Таблица 5. Размещение TEDS в FLASH/EE памяти данных

TEDS	Адрес в FLASH/EE[9]	Размер TEDS
Meta	0x00	76 байт
Channel 1	0x13	96 байт
Channel 2	0x2B	96 байт
	Всего:	268 байт

Существует функция, определенная для загрузки каждой TEDS из пространства программ в область памяти FLASH/EE.

Каждая функция определяет вначале специфичную структуру данных TEDS[10], затем назначает указатель, так что он может быть доступен в виде массива байтов. Рассчитывается контрольная сумма указанного набора данных и автоматически заполняется в поле «контрольная сумма» TEDS. TEDS записывается в FLASH/EE (вызов `TEDS_WriteTEDSToFlash()`). Запомните, что адрес TEDS в FLASH/EE памяти данных, указывающий на записанную TEDS, должен быть определен и передан в этот вызов функции.

```
boolean TEDS_SetupMetaTEDS(void) {
      stMetaTeds idata MetaTEDS = {
      (U32L)sizeof(stMetaTeds)- sizeof(U32L),   // Meta-TEDS length
      2,                                        // 1451 Working Grp Num
      1,                                        // TEDS Version Num

      // Universally unique ID:
      // This UUID represents: 53 deg 30 minutes North,
      // -------------------- 08 deg 0 minutes West,
      // Manufacturers Code: 1,
      // Year: 1999,
      // Time: 00:00:00 on May 4th.
      //

      0x97,0x82,0xC0,0x1C,0x20,0x05,0xF3,0xD0,0x59,0x00,
      0,                                        // Calib TEDS ext key
      0,                                        // NonVolatile Data ext key
      0,                                        // Industry TEDS ext key
      0,                                        // End User App-Spec ext key
      NUM_CHANNELS,                             // Num of implemented chans
      2,                                        // Meta Chan Data Model Len
      0,                                        // Meta Chan Data Reps
      0L,                                       // CH_0 writable TEDS len
      0.0025,                                   // Meta Chan Update Time
      0.001,                                    // Global Write Setup Time
```

[9] Каждое размещение адреса занимает 4 байта.

[10] Данные, представленные здесь являются корректными для реализации.

```
        0.001,                                  // Global Read Setup Time
        0.0005,                                 // Chan Samp period
        0.0005,                                 // Chan Warmup Time
        0.5,                                    // Command Response Time
        0.0003,                                 // STIM handshake Time
        0.04,                                   // EOF Detection Latency
        0.03,                                   // TEDS hold off Time
        0.03,                                   // Operational hold off Time
        2000000,                                // Max Data Rate
        0,                                      // Chan Grp Data Sub-Block

        //
        // суб-блоки группирования каналов отсутствуют
        // => нет повтора полей 25-28.
        //

        0                                       // Checksum (not filled yet)
        };

        // Пространство данных TEDS Data должно программироваться побайтно
        // => мы должны записывать структуру TEDS как массив байтов

        unsigned char *pTEDSByteArray = &MetaTEDS;

        // Расчет контрольной суммы для Meta TEDS, и запись в структуру
        // Meta-TEDS.
        //

        MetaTEDS.Checksum = DAT_CalcChecksum(pTEDSByteArray);

        // Запись структуры Meta-TEDS в FLASH/EE память.
        //

        return TEDS_WriteTEDSToFlash( META_TEDS_ADDRESS,
                                      pTEDSByteArray,
                                      MetaTEDS.Length+sizeof(U32L) );
}
```

Вместе с функциями настройки Channel TEDS (`TEDS_SetupCh1TEDS()` и `TEDS_SetupCh2TEDS()`), это все, что необходимо для установки TEDS на этой системе.

Для получения данных TEDS, обратно из FLASH/EE памяти[11], необходима другая функция. При чтении TEDS из пространства FLASH/EE, она должна быть сохранена в ОЗУ. Поэтому, буфер ОЗУ, который достаточно большой[12] для TEDS, должен быть зарезервирован и передан этой функции. Эти функции разработаны модульно, для того чтобы предоставить расширяемость при изменении кода.

Запомните, что если приложение пользователя отображает TEDS во внешней EEPROM[13] (в отличие от внутренней FLASH/EE), необходимо только изменить один вызов функции в `TEDS_Setup…()` на вызов `TEDS_WriteTEDSToEEPROM()`. Конечно, соответствующая функция должна быть написана.

[11] Когда TEDS считываются NCAP, они индивидуально читаются из FLASH/EE и передаются в ОЗУ побайтно. См. функции `DAT_Read…TEDS()`

[12] Так как существует только 256 байт ОЗУ, буфер должен сосуществовать с данными во время выполнения и стеком. Максимальный размер доступной памяти 150 байт.

[13] Внешняя EEPROM может использоваться в ситуации, когда имеется большее число TEDS, чем может поместиться в FLASH/EE памяти данных. В этом случае, другим решением может стать внесение TEDS только для чтения в FLASH/EE память программ. В этом случае, необходимы минимальные изменения в коде для доступа к переотображенным TEDS.

ПРИЛОЖЕНИЕ 2

ПРИМЕР ДОБАВЛЕНИЯ META-ID TEDS В ЗАГРУЖЕННУЮ ПРОГРАММНУЮ РЕАЛИЗАЦИЮ

См. раздел 5.4 «Meta-Identification TEDS Data Block» стандарта 1451.2.
Единственными модулями, в которые нужно внести изменения, являются «teds.h», «teds.c» и «stim.c».

Изменения в «teds.h»

- должны быть добавлены следующие определения, для описания Meta-Id TEDS. Эти определения описывают «число языков», и существующие «перечни языков» (см. разделы 3.3.7.1 – 3.3.7.3 IEEE 1451.2), поддерживаемые в Meta-Id TEDS.

```
#define TEDS_LANGUAGES     1
#define ENGLISH            25
#define ASCII              10
#define ONE_OCTET          0
```

- следующие определения описывают длину строк, используемых в структуре данных Meta-Id TEDS, и соответственные строки:

```
#define MANU_ID_LEN        16
#define MANU_ID_STRING     "Analog Devices "
#define MODEL_NUM_LEN      10
#define MODEL_NUM_STRING   "ADuC812 v1"
#define VERSION_LEN        10
#define VERSION_STRING     "Demo Ver "
#define SERIAL_NUM_LEN     6
#define SERIAL_NUM_STRING  "123456"
#define DATE_CODE_LEN      6
#define DATE_CODE_STRING   "130599"
#define PROD_DESC_LEN      20
#define PROD_DESC_STRING   "STIM EVAL-ADuC812QS "
```

- определить структурированный тип данных, который описывает «Identification Related Data Sub-Block», суб-блок данных, сопровождающий идентификацию (например, суб-блок, указывающий язык), который будет содержаться в Meta-Id TEDS

```
typedef struct {
   U16L        LangSubBlkLength;
   LANG        StringSpec;
   U8L         ManuIdLength;
   STRING      ManuId[MANU_ID_LEN];
   U8L         ModelNumLength;
   STRING      ModelNum[MODEL_NUM_LEN];
   U8L         VersionCodeLength;
   STRING      VersionCode[VERSION_LEN];
   U8L         SerialNumLength;
   STRING      SerialNum[SERIAL_NUM_LEN];
   U8L         DataCodeLength;
   STRING      DataCode[DATE_CODE_LEN];
   U16L        ProdDescLength;
   STRING      ProdDesc[PROD_DESC_LEN];
} stMetaIDTedsIdDataSubBlk;
```

- определить структурированный тип данных Meta-Id TEDS

```
typedef struct {
   U32L  Length;
   U8C   NumLangs;
   U8E   LangCodeList[TEDS_LANGUAGES];

   // Identification Related Data Sub-Block
   // суб-блок данных, сопровождающий идентификацию

   stMetaIDTedsIdDataSubBlk LangData[TEDS_LANGUAGES];

   // Запомните, что суб-блок будет повторяться
   // в каждом внедренном языке. Мы вводим один язык - английский
   // Channel Grouping Data Sub-Block
   //

   U16L  ChanGroupDataSubBlk;          // поле 18

   // В этой реализации нет суб-блока группирования каналов
   // => не будет повтора полей 19-21
   // Data Integtrity Data Sub-Block
   //

   U16C LanguageBlkChecksum;           // поле 22

   // Data Integrity Data Sub-Block
   //

   U16C Checksum;                      // поле 23
} stMetaIdTeds;
```

- добавить прототип функции «TEDS_SetupMetaIdTEDS()»

Изменения в «teds.c»

- добавить определение расположения Meta-Id TEDS в FLASH/EE памяти данных

```
#define METAID_TEDS_ADDRESS 0x43
```

- добавить новую функцию «TEDS_SetupMetaIdTEDS()», которая будет устанавливать Meta-Id TEDS в FLASH/EE память по ее адресу.

```
boolean TEDS_SetupMetaIdTEDS(void)
{
   // Определение содержимого MetaID-TEDS
   // ---------------------------------------------------

   stMetaIdTeds idata MetaIdTEDS = {

   // Data Structure Related Data Sub-Block
   // -------------------------------------

   (U32L)sizeof(stMetaIdTeds)- sizeof(U32L), // MetaID-TEDS length
   TEDS_LANGUAGES,                           // Num of Langsuages
   ENGLISH,                                  // Languages. . .

   // Identification Related Data Sub-Block
```

```
    // -----------------------------------
    // Lang Sub Blk Len

    (U16L)sizeof(stMetaIDTedsIdDataSubBlk)-sizeof(U16L),
    ASCII,                                  // LANG1: Char Set
    ONE_OCTET,                              // LANG2: Char Code Format
    ENGLISH,                                // LANG3: Str Lang Code
    MANU_ID_LEN,                            // Manu Id Len
    MANU_ID_STRING,                         // Manufacturer Id
    MODEL_NUM_LEN,                          // Model Num Length
    MODEL_NUM_STRING,                       // Model Num
    VERSION_LEN,                            // Ver Code Length
    VERSION_STRING,                         // Ver Code
    SERIAL_NUM_LEN,                         // Serial # Length
    SERIAL_NUM_STRING,                      // Serial #
    DATE_CODE_LEN,                          // Date Code Length
    DATE_CODE_STRING,                       // Date Code
    PROD_DESC_LEN,                          // Product Desc Len
    PROD_DESC_STRING,                       // Product Description

    // Channel Grouping Data Sub-Block
    // -------------------------------
    0,                                      // Chan Grp Data Sub Blk Len
    // Data Integrity Data Sub-Block (1)
    // ---------------------------------
    0,                                      // Checksum for Lang Sub-Blk
    // Data Integrity Data Sub-Block (2)
    // ---------------------------------
    0                                       // Checksum for MetaId TEDS
    };

    // Пространство данных TEDS должно быть запрограммировано побайтно.
    // Поэтому, необходимо читать структуру TEDS как массив байтов.
    //

    unsigned char *pTEDSByteArray = &MetaIdTEDS;

    // Расчет Контрольной суммы для секции Language Sub-Block
    // Meta ID Teds, и заполнение ее в Meta ID TEDS
    // 'data integrity data sub-block (1).
    //

    MetaIdTEDS.LanguageBlkChecksum = TEDS_CalcLangSum(
          MetaIdTEDS.LangData[0].LangSubBlkLength + sizeof(U16L),
          (unsigned char *)&MetaIdTEDS.LangData[0] ) ;

    // Расчет Контрольной суммы для всей Meta-ID TEDS,
    // и заполнение ее в структуру Meta ID TEDS.
    //

    MetaIdTEDS.Checksum = DAT_CalcChecksum(pTEDSByteArray);

    // Программирование Meta-TEDS структуры TEDS в FLASH/EE память данных.
    //

    return TEDS_WriteTEDSToFlash( METAID_TEDS_ADDRESS,
                                  pTEDSByteArray,
                                  MetaIdTEDS.Length+sizeof(U32L) );
}
```

- Заметьте, что в приведенной функции имеется две контрольных суммы, одна для «language sub-block» и одна для всей «Meta-Id TEDS». Функция расчета контрольной суммы TEDS (DAT_CalcChecksum()) работает только для последнего расчета контрольной суммы. Поэтому, локальная функция (DAT_CalcLangSum()) должна быть определена для расчета определенного числа байтов в контрольной сумме.

```
U16C TEDS_CalcLangSum(U16L NumBytes, unsigned char *pStartOfLangBlk)
{
   U16C Offset=0;
   U16C ChkSum=0;
   for(Offset=0; Offset<NumBytes; Offset++)
   {
         ChkSum += *(pStartOfLangBlk+Offset);
   }

   // Настоящая контрольная сумма является дополнением => инверсия разрядов.
   //

   return ~ChkSum;
}
```

- Не забудьте добавить прототип функции DAT_CalcLangSum()в начало файла.

- В функции TEDS_GetTEDSHandle() добавьте следующее условие выбора:

```
case(TEDS_META_ID): ucTEDSAddress = METAID_TEDS_ADDRESS;
break;
```

Изменения в «stim.c»

- в секции инициализации функции «main» (см. рис.6), добавьте вызов функции TEDS_SetupMetaIdTEDS()

Это все, что необходимо для внесения требуемых изменений. Теперь остается только перестроить код и переслать его на макетную плату, как описано в 2.4 и 2.5.

Это код, вместе с примером из пункта 3.2 этого технического замечания, можно загрузить с сайта http://ww.analog.com/microconverter/example1